ZADANIA C++: programowanie obiektowe

Zad. 1:
Skoro znamy konstruktory, to doczytaj o destruktorach.

Zad. 2:
Stwórz klasę *„Liczba zespolona”* i oprogramuj jej działanie zgodnie z regułami matematyki. Powinna posiadać pary metod `get-set`, trzy konstruktory i być rozdzielona do odpowiednich plików .h i .cpp.

Zad. 3:
Napisz klasę realizującą rozwiązywanie równania kwadratowego.
Klasa powinna posiadać pola prywatne `A`, `B`, `C`, reprezentujące współczynniki równania, wraz z odpowiednimi akcesorami i modyfikatorami.
Dodatkowo należy stworzyć funkcję składową `delta()`, obliczającą wyróżnik trójmianu oraz trzy funkcje obliczające miejsca zerowe.
Klasa powinna być wyposażona w konstruktor domyślny oraz trójparametrowy konstruktor ogólny.
Należy zastanowić się nad przypadkiem, gdy równanie staje się liniowym tzn. `A = 0` i zaproponować rozwiązanie tego problemu.

Obiekt stworzonej klasy może być wykorzystany następująco:

```
RownanieKwadratowe r;
double num;
cout << 'Podaj A:'; cin >> num; r.ustawA( num );
cout << 'Podaj B:'; cin >> num; r.ustawB( num );
cout << 'Podaj C:'; cin >> num; r.ustawC( num );
if( r.delta() > 0 )
     cout << "Pierwiastki rownania x1=" << r.obliczX1() <<
     " x2=" << r.obliczX2() << endl;
else
if(r.delta() = 0 )
     cout << "Pierwiastek podwójny x12=" << r.obliczX12()
     << endl;
else
     cout << "Brak pierwiastkow rzeczywistych" << endl;
```

Obiekty mogą być inicjowane następująco:

```
RownanieKwadratowe r1; // Konstruktor inicjuje A=B=C=0;
RownanieKwadratowe r2(3,2,1); // Konstruktor inicjuje A=3,B=2,C=1;
```

Zad. 4:
Napisz klasę `Macierz` i zaimplementuj na jej rzecz przeciążenie operatorów tak, aby realizowały następujące funkcje:
+ dodawanie macierzy,
- odejmowanie macierzy,
* mnożenie macierzy przez skalar.

Zad. 5:

Załóżmy, że gromadzimy wodę mineralną. Są trzy rodzaje butelek – o pojemności 2l (duże), średnie o pojemności 1l oraz małe o pojemności 0.5 litra. Stworzyć klasę MyWater z metodami:
- `void addLarge(int)` – dodaje do zapasu wody podaną jako argument liczbę dużych butelek.
- `void addMedium(int)` – dodaje do zapasu wody podaną jako argument liczbę średnich butelek.
- `void addSmall(int)` – dodaje do zapasu wody podaną jako argument liczbę małych butelek.

Dodatkowo należy zaimplementować metody umożliwiające uzyskanie informacji o tym, ile jest każdego rodzaju butelek, oraz jaka jest łączna pojemność zgromadzonej wody.
Pojemności butelek (dużych, małych, średnich) przedstawić jako pola statyczne. Dostarczyć metod pozwalających uzyskiwać informacje o tych pojemnościach oraz je zmieniać.
Klasę przetestować wyprowadzając dane na ekran:
- Mam teraz 6.5 litrów wody.
- Dużych butelek: 2
- Małych butelek: 3
- Średnich butelek: 1

Zad. 6:

Napisz program, w którym zdefiniujesz jedną z wymienionych hierarchii:
1. Pojazdów
2. Książek
3. Co Ci do głowy wpadnie :-)

Zadbaj o poprawność schematu dziedziczenia, odpowiednie zadeklarowanie pól i metod w zależności od poziomu szczegółowości, itp. Niezależnie od wszystkiego – prowadź ewidencję, ile obiektów danej klasy aktualnie istnieje.

Zad. 7:

1. Wykonaj prostą grę polegającą na symulacji zachowań drogowych.
2. Zdefiniuj klasy użytkowników drogi (pieszych, samochodów, rowerów).
3. Zadbaj o ich dziedziczenie z abstrakcyjnej klasy bazowej.
4. Każdy z obiektów danej klasy potrafi się poruszać po planszy (konsoli) z różną prędkością. Każdy z obiektów ma również swój odpowiedni znaczek.
5. Obiekty poruszają się po planszy w sposób losowy, gra kończy się w momencie „kraksy”.
6. Zadbaj o krokową pracę (aby można było podglądać każde stadium).
7. Przeprowadź symulacje dla różnej wielkości planszy oraz dla różnej liczby użytkowników drogi.

Zad. 8:

Napisz klasę o nazwie `Motorowka`, której składnikami będą cena, moc i maksymalna prędkość. Dopisz również dwie funkcje: pobierającą i wypisującą parametry motorówki. Dodaj prywatne atrybuty dla wektora prędkości. Napisz dwie metody zmieniające wektor prędkości motorówki: jedną, nadająca jej zupełnie nowy wektor prędkości, oraz drugą, zachowującą kierunek, ale zmieniającą prędkość (długość wektora). Przyjmij, że motorówka nie pływa do tyłu. W funkcji głównej wywołaj poszczególne metody.

Zad. 9:

Zdefiniuj klasę `Kolo`, która będzie zawierać:

- pola prywatne:
  - promień koła (typ `float`),
  - kolor wypełnienia (typ `int`)
- konstruktor bezparametrowy inicjujący wartość początkową pola koła (dowolnie ustaloną),
- konstruktor z parametrami promień, kolor wypełnienia,
- metody publiczne:
  - `Oblicz_Pole(..)`, obliczająca pole powierzchni koła,
  - `Ustaw_Promien(..)`, zmieniająca promień koła,
  - `Ustaw_Kolor(..)`, zmieniającą kolor koła.

Zdefiniuj przeciążenie operatora == porównujące, czy dwa koła mają taką samą powierzchnię i kolor.

Uruchom w programie głównym sekwencję instrukcji dla dwóch obiektów `K1`, `K2`: konstruktor bezparametrowy, porównanie pól, ustawienie tego samego koloru, porównanie pól, ustawienie różnych kolorów, porównanie pól. Użyj na końcu konstruktora z parametrami dla obiektów `K3`, `K4` i pokaż różne warianty ich porównania.

Dodatkowo:

Spróbuj wykonać to samo z udziałem wskaźników na obiekty oraz uwalnianiem pamięci dynamicznej.

Zad. 10:

Stwórz klasę `Osoba`, która będzie mieć `Imie` i `Plec`. Stwórz klasę `Student`, która będzie dziedziczyć po klasie `Osoba` i dodatkowo będzie mieć daną składową `Punktacja`. Zaimplementuj metodę `wyswietlInformacje()`, która dla klasy `Osoba` wypisze imię i płeć. Niech klasa `Student` skorzysta z metody klasy nadrzędnej, a dodatkowo wypisze informacje nt. punktacji. Stwórz w klasie wspólny licznik wszystkich utworzonych obiektów.

W funkcji `main()` zademonstruj działanie wszystkich klas.

Dodatkowo:

Stwórz klasę `Punktacja`, która będzie posiadała trzy dane składowe: `maksymalnaPunktacja`, `otrzymanaPunktacja`, `procent`. Zastąp tą klasą odpowiednie dane składowe klasy `Student`.

Zad. 11:

Zaprojektuj klasę reprezentującą `Samolot`. Zaproponuj atrybuty dla tej klasy pozwalające na określenie jego rozmiaru i maksymalnej liczby pasażerów oraz wektora prędkości (składa się on z kierunku oraz prędkości). Atrybuty dla wektora prędkości mają być prywatne. Napisz dwie metody zmieniające wektor prędkości statku: jedną, nadająca mu zupełnie nowy wektor prędkości, oraz drugą, zachowującą kierunek, ale zmieniającą prędkość (długość wektora). Pamiętaj, że samolot nie lata do tyłu.

W funkcji `main()` zademonstruj działanie klasy.

Zad. 12:
Zdefiniuj klasę `Prostokat`, która będzie zawierać:

- pola prywatne typu całkowitego:
  - długości boków prostokąta,
  - kolor wypełnienia
- konstruktor bezparametrowy inicjujący wartość początkową pola prostokąta (dowolnie ustaloną),
- konstruktor z parametrami długość, wysokość i kolor wypełnienia prostokąta,
- metody publiczne:
  - `Daj_Dlugosc(..)`, zwracającą długość prostokąta,
  - `Daj_Wysokosc(..)`, zwracającą wysokość prostokąta.

Zdefiniuj przeciążenie operatora == porównujące, czy dwa prostokąty mają boki tej samej długości (mogą być podane w odwrotnej kolejności!) i kolor.

Uruchom w programie głównym sekwencję instrukcji dla dwóch obiektów `P1`, `P2`: konstruktor bezparametrowy, porównanie czy są takie same. Użyj też konstruktora z parametrami dla obiektów `P3`, `P4` oraz `P5`, `P6` i pokaż różne warianty porównania ich ze sobą.

Zad. 13:
1. Zdefiniować klasy reprezentujące płaskie figury geometryczne: `Kwadrat`, `Prostokat`, `Kolo`, `Trapez`. Klasy powinny posiadać pola przechowujące informacje niezbędne do obliczenia pól tych figur.
2. Pola powinny być prywatne, dostęp do nich powinien być zapewniany przez odpowiednie metody pobierania i ustawiania wartości. Klasy powinny posiadać konstruktor bezparametrowy oraz ogólny, zapewniające inicjalizację wszystkich pól.
3. Obliczenie pola danej figury powinna realizować odpowiednia funkcja składowa o nazwie `obliczPole` — wyznaczona wartość pola powinna być jej rezultatem.

Zad. 14:
Bazując na klasach opisu figur płaskich stworzonych wcześniej, zdefiniować ich klasy pochodne, reprezentujące bryły: sześcian, prostopadłościan, kula, graniastosłup o podstawie trapezu. W klasach pochodnych należy dodać wszelkie informacje konieczne dla obliczenia pól tych brył oraz należy przedefiniować funkcje składowe obliczania pola (funkcje `obliczPole` każdej z klas reprezentujących figurę płaską), tak by wyznaczały właściwe pola brył.

Zad. 15:
Zdefiniuj klasę napis. Zaimplementuj przeciążenie operatorów + oraz +=.

Zad. 16:
Zdefiniuj klasę napis. Zaimplementuj przeciążenie operatorów - oraz -=.

Zad. 17:

Stwórz książkę adresową do przechowywania danych osób, opartą na obiektach. Program ma posiadać następujące menu:

1. Dodaj nową osobę.
2. Edytuj/usuń osoby.
3. Wyszukaj osoby.
4. Wyświetl osoby.
5. Importuj dane z pliku.
6. Eksportuj dane do pliku.
7. Zakończ program.

Dodaj nową osobę umożliwia dodanie nowej osoby do bazy danych na podstawie parametrów takich jak: Imię i nazwisko, data urodzenia, pełny adres, oraz opcjonalnie: numer gg, adres e-mail, adres www.

Edycja powoduje wyświetlenie wszystkich dostępnych osób i na podstawie tej listy ma być możliwość wyboru konkretnej osoby do edycji lub do usunięcia. Usunięcie osoby ma być potwierdzane komunikatem *„Czy jesteś pewien?”*.

Wyszukiwanie osób ma być możliwe po nazwisku, dacie urodzenia lub mieście zamieszkania.

Wyświetl osoby – wyświetla wszystkie osoby z książki z możliwością sortowania według imienia, nazwiska, roku urodzenia lub miasta.

Import i eksport kolejno wczytują lub zapisują dane do pliku tekstowego.

Ma być też możliwość wyświetlenia liczby zapisanych osób. Dodatkowo każda osoba powinna mieć swój unikalny identyfikator.

Każda osoba to obiekt danej klasy. Książka adresowa agreguje obiekty klasy osoba. Należy wykorzystać pola i 1 metodę statyczną, co najmniej 2 konstruktory oraz pamiętać o hermetyzacji.

Zad. 18:

Napisać klasę `Zamowienie`, która umożliwia przechowanie i zarządzanie informacjami związanymi z zamówieniem na towar. Klasa ta powinna zawierać następujące **pola prywatne**:

- `int * towary` – tablica z listą towarów w zamówieniu (towar reprezentowany przez identyfikator będący liczbą całkowitą),
- `double * ceny` – tablica z cenami poszczególnych towarów w zamówieniu (każda cena przypisana jest do towaru zapisanego pod tym samym indeksem w tabeli),
- `int maxPoz` – maksymalna liczna pozycji - zamówienie nie może mieć więcej niż `maxPoz` pozycji (towarów),
- `int ilePoz` – aktualna liczba pozycji - przechowuje aktualną liczbę pozycji zamówienia.

Konstruktory:

- Konstruktor **ogólny** z parametrem (`int maxPoz`) określającym maksymalną liczbę pozycji w zamówieniu. Konstruktor powinien nadać odpowiednie wartości wszystkim polom klasy i pozwolić na stworzenie tablic (towary, ceny).
- Konstruktor **kopiujący**, umożliwiający stworzenie zamówienia na podstawie już istniejącego.

Metody publiczne:

- `void dodajPozycje(int idTowaru, double cenaTowaru)` – funkcja dodaje towar do listy elementów zamówienia, jeżeli jest jeszcze na niej miejsce.
- `int najdrozszyTowar()` – funkcja zwraca identyfikator towaru (z tablicy towary), którego koszt jest największy. W przypadku, gdy zamówienie jest puste wynikiem jest -1.
- `double kosztCalkowity()` – funkcja zwraca sumaryczny koszt zamówienia.
- `void wyswietl()` – funkcja wyświetla całość zamówienia, a więc identyfikatory wszystkich towarów i odpowiadające im ceny oraz koszt całego zamówienia.

Zad. 19:

Poniżej podana została klasa `Lista`, która służy do przechowywania listy liczb całkowitych, uzupełnij metody tej klasy.

```
class Lista {
private:
      int liczby[100]; // tablicy na przechowywane liczby
      int pojemnosc; // liczba elementów, które można // pomieścić na liście
      int rozmiar; // liczba elementów aktualnie // przechowywana na liście
public:
      Lista();
      // dodaje nową liczbę do listy
      void dodaj(int liczba);
      // zwraca element listy znajdujący sie na podanym indeksie
      int dajElement(int indeks);
      // wyświetla aktualny rozmiar listy, tj. liczbę elementów
      int wezRozmiar();
};
```

Przykład użycia klasy lista:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "Lista.h"
using namespace std;
int main()
{
      Lista l;
      srand(time(NULL));
      for(int i = 0; i < 20; i++)
            l.dodaj( rand() % 100 );
      cout << "Zawartosc listy: ";
      for(int i = 0; i < l.wezRozmiar(); i++)
            cout << l.dajElement(i) << " ";
      cout << endl;
      return 0;
}
```

Zad. 20:

Napisać dwie klasy, których celem będzie liczenie **pola** i **objętości** następujących figur:

- Kwadrat
- Sześcian

Każda klasa powinna posiadać dwa konstruktory: **domyślny** i **ogólny**, które odpowiednio zerują wartości pól klasy lub ustawiają je na wartości podane podczas tworzenia obiektu. Klasa sześcian powinna dodatkowo zawierać konstruktor kopiujący i rzutujący. Proszę zadbać o to, by możliwa do ustalenia **długość boków** była większa od **zera** – nie można ustawić np. kwadratu o długości boku zero. Jeżeli jednak użytkownik będzie próbował ustawić taką długość boku, proszę tej wartości nie akceptować (nie przypisywać do pola), a przy próbie wyliczenia objętości lub pola powierzchni danej figury wyświetlać komunikat *„Operacja niemożliwa do zrealizowania. Sprawdź ustawioną długość boków”*. Niedozwolone jest deklarowanie pól klasy w sekcji `public`.

Zad. 21:

Zdefiniuj klasę `Przedmiot` reprezentującą przedmiot ćwiczeniowy studenta. Klasa powinna zawierać:

- pola prywatne:
- `double oceny[];` // oceny studenta
- `int nr_indeksu;` // numer indeksu studenta
- `int liczba_ocen;` // liczba wprowadzonych ocen
- `int limit_ocen;` // maksymalna liczba ocen

- publiczny konstruktor z dwoma parametrami:
- `int nr_indeksu, int limit_ocen`

- metody publiczne:
- `double obliczSrednia();` // oblicza średnią arytmetyczną ocen
- `bool zaliczenie();` // określa, czy student otrzyma zaliczenie
- `void dodajOcene(double ocena);` // dodaje nową ocenę
- `void zmienOcene(double ocena, int id);` // zmienia ocenę
- `void pisz();` // wypisuje na ekranie informację

Dodatkowe informacje:

- Funkcja `obliczSrednia` oblicza i zwraca średnią arytmetyczną wpisanych już ocen.
- Funkcja `zaliczenie` ustala, czy student otrzyma zaliczenie (średnia wszystkich uzupełnionych ocen większa lub równa 3.0). Jeśli otrzyma, to zwraca `true`, w przeciwnym razie `false`.
- Funkcja `dodajOcene` dodaje nową ocenę (jeśli jest to możliwe – nie przekroczy limitu).
- Funkcja `zmienOcene` zmienia ocenę na podanej pozycji (`id` liczonej od 0), ale tylko jeśli taka ocena istnieje.
- Funkcja `pisz` wyświetla na ekran studenta, wszystkie uzupełnione oceny i informację, czy aktualnie dostałby zaliczenie (*tak* lub *nie*).

Zad. 22:

Mamy daną klasę:

```
class wektor    {
public:
    double x, y, z;
    wektor (double xx = 0, double yy = 0, double zz = 0){
        x = xx;
        y = yy;
        z = zz;
    };
};
```

Przeciążyć operatory, aby realizowały następujące funkcje:

+ dodawanie wektorów,
- odejmowanie wektorów,
* iloczyn skalarny.

Sprawdź iloczyn skalarny na następującym przykładzie: (1, -3, -5) + (4, 3, -1) = 0.

Zad. 23:

Stwórz klasę reprezentującą liczby wymierne. Zaimplementuj podstawowe działania arytmetyczne przeciążenie operatorów wejścia/wyjścia (tj. << oraz >>).

Zad. 24:
Utwórz klasę reprezentującą przedziały otwarte liczb rzeczywistych. Zaimplementuj funkcje wyznaczania: części wspólnej, arytmetycznej sumy i iloczynu przedziałów.

Zad. 25:
Stwórz klasy reprezentujące: punkt, odcinek i prostą. Zapewnij metody pozwalające znajdować odległość punkt od innego punktu oraz prostej, a także metodę wyznaczająca długość odcinka.

Punkt: A = (xa, ya)

Punkt: B = (xb, yb)

Prosta: $c \cdot x + d \cdot y + e = 0$

Odległość punktów od siebie: $\sqrt{(xb-xa)^2+(yb-ya)^2}$

Odległość punktu A od prostej: $\frac{|c \cdot xa + d \cdot ya + e|}{\sqrt{c^2+d^2}}$

Zad. 26:
Stwórz wzorzec klasy zawierający wspólny licznik wszystkich obiektów utworzonych z wszystkich klas konkretyzowanych z tego wzorca.

Zad. 27:
Stwórz typ `para` zawierający publiczne pola `x` oraz `y` typu `double` (jako klasę albo jako strukturę). Typ ten reprezentuje parę liczb rzeczywistych, traktowaną zależnie od kontekstu jako punkt bądź wektor.

Stwórz klasę okręgu, tak aby obiekty tej klasy można było stworzyć na każdy z poniższych sposobów:

- podając środek oraz promień okręgu,
- podając środek oraz punkt leżący na okręgu,
- podając dwa przeciwległe punkty na okręgu,
- podając trzy punkty leżące na okręgu,
- podając trzy punkty stanowiące wierzchołki trójkąta opisanego na tym okręgu.

Zad. 28:
Zdefiniuj typ `wektor4` jako wektor czterech liczb rzeczywistych. Zdefiniuj dla tego typu operatory: +, -, *, /, =, +=, -=, *= i /= dla kombinacji wektorów i liczb rzeczywistych.
http://pl.wikipedia.org/wiki/Wektor

Zad. 29:
Zdefiniuj klasę `wektor`, w której rozmiar będzie argumentem konstruktora `wektor::wektor(int)`. Zdefiniuj dla tego typu operatory: +, -, *, /, =, +=, -=, *= i /= dla kombinacji wektorów i liczb rzeczywistych.
http://pl.wikipedia.org/wiki/Wektor

Zad. 30:
Zdefiniuj:
```
class podstawowa {
public:
     virtual void jaJestem() {
          cout << "podstawowa\n";
     }
};
```
Wyprowadź z klasy podstawowa dwie klasy pochodne i w każdej zdefiniuj funkcję `jaJestem()`, wypisującą nazwę klasy. Utwórz obiekty tych klas i wywołaj na nich `jaJestem()`. Przypisz adresy obiektów klas pochodnych wskaźnikom typu `podstawowa*` i – korzystając z tych wskaźników – wywołaj `jaJestem()`.

Zad. 31:
Zdefiniuj klasę `trojkat` i klasę `okrag`. Obie klasy muszą zawierać klasę `figura` jako publiczną klasę podstawową.

Zad. 32:
1. Stworzyć dwie klasy. Każda ma po 3 pola, które są liczbami naturalnymi. Dwa z nich są polami prywatnymi.
2. Zainicjować po obiekcie danej klasy. Wyświetlić w wątku głównym wszystkie pola (stworzyć potrzebne metody).
3. Stworzyć metodę, której atrybutem będzie obiekt drugiej klasy. Metoda ma dodać do każdego pola obiektu swoje pierwsze pole, podzieli przez drugie (jeśli zero, obsłużyć wyjątek), a następnie zapiszę wyniki tych działać w pola obiektu, który został jej przekazany.

Zad. 33:
Napisz definicje instrukcji inicjujących do poniższej klasy:
```
class Vector
{
     private:
          double x;
          double y;
     public:
          Vector();
          Vector(double, double);
};
```
Klasa ma reprezentować wektor w przestrzeni dwuwymiarowej, a instrukcje inicjujące mają realizować inicjalizację tego wektora. Pierwsze instrukcje inicjujące powinny ustawiać wektor na wartość domyślną (0,0).

Dopisz konstruktor kopiujący.
```
Vector(const Vector&);
```

Zad. 34:
Napisz obiektową wersję gry Kółko i krzyżyk. *Wskazówki*: dobrym kandydatem na obiekt jest oczywiście plansza. Zdefiniuj też klasę graczy, przechowującą ich imiona (niech program pyta się o nie na początku gry).

Zad. 35:

Poniższa klasa miała implementować dowolnej wielkości tablicę obiektów klasy Vector z poprzedniego ćwiczenia. Niestety okazało się, że powoduje wycieki pamięci – programista zapomniał o napisaniu destruktora:

```
class VectorsArray
{
     public:
          Vector* vectors;

          VectorsArray(size_t);
          Vector GetVector(size_t);
          size_t GetSize();
          size_t size;
};

VectorsArray::VectorsArray(size_t argSize)
: size(argSize)
, vectors(new Vector[argSize])
{
}
Vector VectorsArray::GetVector(size_t i)
{
     return vectors[i];
}
size_t VectorsArray::GetSize()
{
   return size;
}
```

Do powyższej klasy dopisz definicję destruktora. Nie zapomnij o dealokacji pamięci!

Zad. 36:

Zaimplementuj listę jednokierunkową jako klasę.

Zad. 37:

Zaimplementuj listę dwukierunkową jako klasę.

Zad. 38:

Zaimplementuj drzewo przeszukiwań binarnych BST jako klasę.

Zad. 39:

Utwórz klasę `Plik`. Klasa powinna zawierać:

- pole prywatne - wskaźnik na zmienną plikową typu `FILE`,
- konstruktor z parametrem otwierający plik dyskowy o podanej nazwie - wskaźnik na nazwę,
- destruktor zamykający plik dyskowy.

Uruchom program z obsługą obiektową pliku.

Zad. 40:

Napisz program do prowadzenia dokumentacji i wykonywania analiz statystycznych dla klasy 20 uczniów. Informacja o każdym uczniu zawiera: ID, imię, płeć, wyniki kolokwiów (3 kolokwia w semestrze) i wynik końcowy (średnią z kolokwiów). Użyj do tego celu klasy – zaimplementuj odpowiednie funkcjonalności, z których będzie można później skorzystać.
W funckji `main()` program poprosi użytkownika, aby wybrać z menu:

```
==================================================
                        MENU
==================================================
1. Dodaj wpis.
2. Usuń wpis.
3. Aktualizuj wpis.
4. Wyświetl wszystkie wpisy.
5. Oblicz średnią wyników wybranego studenta.
6. Wyświetl średnią wyników wybranego studenta.
7. Oblicz i wyświetl średnią wszystkich studentów.
8. Wyświetl najlepszy i najgorszy wynik (wraz z danymi studentów,
którzy je osiągnęli).
Wpisz swój wybór:
```

Zad. 41:

Utwórz klasę `Punkt`. Klasa powinna zawierać:

- 2 pola prywatne typu całkowitego (współrzędne punktu),
- konstruktor bezparametrowy inicjujący dowolne wartości początkowe punktu,
- konstruktor z parametrami (współrzędne punktu),
- metodę publiczną - zwracającą współrzędną `x`,
- metodę publiczną - zwracającą współrzędną `y`.

Zdefiniuj przeciążenie operatora == porównujące dwa punkty, czy się nakładają.

Uruchom program dla dwóch różnych obiektów, porównaj je, a następnie dokonaj zmian w jednym z nich, by uczynić obiekty identycznymi. Spróbuj wykonać te same działania w wersji z dynamicznym przydziałem pamięci.

Zad. 42:

Zaprojektuj klasę reprezentującą statek. Zaproponuj atrybuty dla tej klasy pozwalające na określenie jego rozmiaru i maksymalnej liczby pasażerów oraz wektora prędkości. Atrybuty dla wektora prędkości mają być prywatne. Napisz dwie metody zmieniające wektor prędkości statku: jedną, nadająca mu zupełnie nowy wektor prędkości, oraz drugą, zachowująca kierunek, ale zmieniająca prędkość (długość wektora). Przyjmij, że statek nie pływa do tyłu.
W funkcji `main()` zademonstruj działanie klasy.

Zad. 43:

Napisz klasę o nazwie `Samochod`, której składnikami będą cena, moc i maksymalna prędkość.
Dopisz również dwie funkcje: pobierającą i wypisującą parametry samochodu. W funkcji głównej wywołaj je.

Zad. 44:

Zaprojektuj klasę `Klient`: dane osobowe klienta – imię, nazwisko, e-mail, nr telefonu.

Zaprojektuj klasę `Rezerwacja`:

1. numer referencyjny;
2. `Klient`;
3. data rezerwacji;
4. data przyjazdu;
5. rodzaj pokoju;
6. liczba nocy;
7. uwagi;
8. rabat.

Dodaj metody dostępowe do zmiennych (pól) klasy.

Utwórz konstruktory pozwalające utworzyć obiekty.

Zwróć uwagę, że wartość pola numer referencyjny rezerwacji klasy `Rezerwacja` jest nadawany w momencie tworzenia obiektu (np. na podstawie znacznika czasu lub inny dowolny ciąg znaków) i później ta wartość nie może być zmieniona. Również jako data rezerwacji jest ustawiana data bieżąca w momencie tworzenia obiektu.

Zabezpiecz, aby nie można było utworzyć rezerwacji dla daty przyjazdu, która już minęła.

Dodaj metodę obliczającą kwotę noclegu na podstawie następujących stawek oraz rabatu:

Rodzaj pokoju:

A 290,00

B 330,00

C 410,00

D 450,00

E 510,00

F 520,00

G 1150,00

Zad. 45:

Zaimplementuj klasę `Ulamek` z prywatnymi polami całkowitymi `Licznik` i `Mianownik`. Należy odpowiednio przeciążyć operatory `*`, `-`, `+`, `*=`, `=`, `-=`, `+=`, `--`, `++`, `<`, `>`, `==`, `<<` (wypisywanie do strumienia) oraz zaimplementować konstruktory. Mianownik nie może być równy zero. Ułamki mają być wypisywane w postaci: `Licznik/Mianownik`, np.: „`3/4`”, „`-79/8`”.

W funkcji `main()` powołaj do życia kilka obiektów oraz zademonstruj działanie przeciążonych operatorów klasy `Ulamek`.

Zad. 46:

Zaimplementuj klasę `Data`, której składowymi prywatnymi będą: `dzien`, `miesiac`, `rok`. Utwórz funkcję `pobierzDate()`, która będzie pobierać datę od użytkownika.

Przeciąż operator << aby wypisywał datę zapisaną w obiekcie klasy. Jeśli dzień lub miesiąc będą jednocyfrowe, to dopisz przed nimi „`0`”.

Stwórz konstruktor bezparametrowy, który domyślnie ustawia datę na 1 stycznia 1970.

Stwórz konstruktory: jeden pobierający tylko rok (pozostałe składniki domyślnie ustawia na 1 stycznia), drugi pobierający tylko dzień i miesiąc (rok domyślnie ustawia na 1970).

Stwórz metody pozwalające na zmianę każdego pola osobno. Możesz z nich skorzystać w konstruktorach.

Zadbaj o poprawność wprowadzanych danych.

W funkcji `main()` zademonstruj działanie zaimplementowanych funkcjonalności.

Zad. 47:

Dany jest układ równań liniowych

$$A_1x + B_1y = C_1$$
$$A_2x + B_2y = C_2$$

Rozwiązanie układu równań może polegać na wyliczeniu odpowiednich wyznaczników W, Wx, Wy a następnie ich iloczynów — zgodnie z informacjami poznanym na zajęciach z matematyki.
Należy zaprojektować i zaimplementować obiektowy program pozwalający na rozwiązywanie dowolnego układu takich równań. Program powinien umożliwiać wczytanie współczynników A_1, B_1, C_1, A_2, B_2, C_2, następnie powinien wyznaczyć rozwiązania równań metodą wyznacznikową. Należy identyfikować i prawidłowo zareagować na sytuację, gdy układ jest nieokreślony.

Zad. 48:

Napisać klasę `Oceny` reprezentującą informacje o ocenach studentów. Klasa powinna zawierać:
• pole prywatne: `float[][] oceny` – tablica z listą ocen studentów (pierwszy wymiar – numer studenta, drugi wymiar – numer przedmiotu),
• konstruktor z dwoma parametrami (`int liczbaStudentów`, `int liczbaPrzedmiotow`) – powinien tworzyć i zainicjować odpowiednie tablice.
Metody publiczne:
• `float ObliczSrednia(int student)` – oblicza i zwraca średnią arytmetyczną podanego studenta dla wpisanych już ocen (brak oceny to -1 w odpowiednim elemencie tablicy),
• `boolean ZaliczonySemestr(int student)` – zwraca `false`, jeśli średnia ocen jest mniejsza od 3.0 lub brak oceny z któregoś z przedmiotów, w przeciwnym wypadku `true`,
• `void WpiszOcene(int student, int przedmiot, float ocena)` – dodaje nową ocenę,
• `void zmienOcene(int student, int przedmiot, float ocena)` – zmienia ocenę na podanej pozycji wpisując średnią oceny podanej i istniejącej, ale tylko jeśli taka jest już wpisana,
• `void Pisz()` – wyświetla na ekran oceny studentów (w jednym wierszu: numer albumu, lista ocen różnych od -1, średnia ocen).

Zad. 49:

Zaimplementuj obiektową wersję funkcji `dzienRoku(dzien, miesiac, rok)`, która pobiera trzy argumenty będące datą, a zwraca, którym dniem roku jest podana data. Zadbaj o idiotoodporność funkcji – jeśli data, z którą została wywołana funkcja, nie istnieje, funkcja zwróci 0.

Pamiętaj o latach przestępnych – rok jest przestępny, jeśli jest podzielny przez 400 lub jest podzielny przez 4 i nie jest podzielny przez 100.

Wykorzystaj tablicę dwuwymiarową – pierwszy wiersz przechowuje liczbę dni miesięcy w roku normalnym, drugi wiersz w roku przestępnym.

W `main()` sprawdź działanie funkcji dla następujących danych:

```
dzienRoku(2014,05,27);   // 147
dzienRoku(2015,13,20);   // 0
dzienRoku(1983,02,29);   // 0
dzienRoku(1984,02,29);   // 60
dzienRoku(2000,02,29);   // 60
dzienRoku(2004,02,29);   // 60
dzienRoku(1900,02,29);   // 0
dzienRoku(2100,02,29);   // 0
```

Zad. 50:

Napisz klasę pozwalającą na obliczenie podatku dochodowego dla osoby fizycznej. Uwzględnij doliczenia i odliczenia od podatku, a także skalę podatkową wg wzoru:
Dochód < 1800 zł -> podatek = 0;
Dochód >=1800 zł oraz dochód <100000 zł -> podatek = 15% dochodu
Dochód >=100000 zł -> podatek = 40% dochodu
Wszystkie pola w klasie mają być prywatne, a do ich ustawiania i odczytywania użyj selektorów i modyfikatorów. Napisz konstruktor, który zeruje pola oraz konstruktor, w którym podajesz dochód oraz doliczenia.
Prototyp:

```
Podatnik kowalski = new Podatnik();
kowalski.setDochod(1587);
kowalski.setDoliczenia(-5);
cout << kowalski.ObliczPodatek() << endl;
Podatnik iksinski = new Podatnik(48765, 120);
cout << iksinski.ObliczPodatek();
```

Zad. 51:

Napisz klasę służącą do analizy częstości występowania znaków w tekście. Program ma nic nie wyświetlać na ekran, a jedynie zapisywać wyniki analizy do pliku tekstowego. Prototyp użycia:

```
Analizator a = new Analizator("testowy.txt");
a.zapisz("testowy");
```

Zad. 52:

Napisz klasę `Serwis` wspomagającą obsługę serwisową notebooków. W konstruktorze domyślnym powinna być zerowana prywatna tablica `naprawy[][]` oraz ustawiane statyczne pole `liczbaZgloszen` na 0. Zakładamy, że pierwszy wymiar tablicy to identyfikatory komputerów (max 100), a drugi – daty napraw (max 5), Dodatkowo napisz metody:

a) `boolean dodajNaprawe(int nrKomputera)`: funkcja ta pobiera aktualne naprawy komputera o indeksie `nrKomputera` (wiersz z tabeli), a następnie w kolejnym wolnym miejscu dopisuje dzisiejszą datę. Jeśli brakuje miejsca na naprawę, funkcja powinna zwrócić `false`; w przeciwnym wypadku `true`.

a) `int liczDoZlomowania()`: funkcja liczy, ile komputerów już nie może być naprawionych (brak miejsca w tablicy na naprawy)

a) `String ostatniaNaprawa(int komputer)`: funkcja ma zwrócić `string`a z datą ostatniej naprawy danego komputera

a) `double średnioNapraw()`: funkcja zwraca informacje o średniej liczbie napraw komputerów w bazie. UWAGA: komputer nigdy nie naprawiany nie wchodzi do średniej!

Zad. 53:

Napisz funkcję `odczytaj()`. Jej funkcjonalnością będzie odczytywanie informacji o osobach zawartych w pliku tekstowym do oddzielnych instancji (obiektów) klasy `Osoba`.

```
Public class Osoba {
     int wiek;
     string imię
     string nazwisko;
     string[] dzieci;
     double dochod;
}
```

Plik wejściowy ma następującą postać:

```
wiek   imię   nazwisko  liczba_dzieci  imiona_dzieci_rozdzielone_tabem    dochód
```

Przykładowo:

```
35   Jan   Kowalski   3    Ania Zdzisio   Joanna         5000
```

Zad. 54:

W pliku **obiektów (nie rekordków, nie tekstowym)** `równania.obj` zapisane są obiekty klasy `Rownanie`:

```
class Rownanie {
     String wyrażenie;
     Boolean czyPoprawne;
}
```

Napisz program, w którym: 1) odczytasz strukturę obiektów do tablicy zadeklarowanej lokalnie (przyjmij, że będzie ich nie więcej niż 100), 2) sprawdzisz poprawność wyrażenia (wyrażenie jest poprawne, jeśli zawiera tyle samo liczb, co liter; inne znaki są pomijane), 3) uzupełnisz pole `czyPoprawne` odpowiednią informacją, 4) zapiszesz obiekty z powrotem do tego samego pliku, z którego odczytałeś nadpisując zawartość, 5) wypiszesz na ekran informację o liczbie poprawnych obiektów.

Zad. 55:

Napisz program „generator losowych numerów telefonów”. Program ma umożliwiać generowanie podanej przez użytkownika liczby numerów telefonów z konkretnej strefy numeracyjnej. Dodaj konstruktor domyślny zerujący wszystkie pola oraz pole statyczne, którego wartość będzie mówić o liczbie obiektów klasy `Telefony` stworzonych do tej pory. Przykładowe zastosowanie:

```
for(int i=11;i++;i<33) {

Telefony Tel = new Telefony();

Tel->setWojewództwo(i); //ustawienie strefy numeracji na i
     oraz wyczyszczenie tablicy wewnętrznej z numerami
     telefonów
Tel->generate(1000); //wygenerowanie 1000 numerów,
     zapisywanych w tablicy (odpowiednio zadeklarowanej
     wcześniej).

Tel->show(); //wypisuje na ekran numery zapisane w tablicy
}

cout << Telefony::ile << endl; //wyświetli... ile?
```

Zad. 56:

Napisz klasę `Pożyczkobiorca`. Będzie ona pomocna przy udzielaniu pożyczki dla konsumentów przez instytucje bankowe. Klasa ma zawierać pola określające imię, nazwisko, wiek i PESEL klienta (do ich ustawiania użyj **enkapsulacji**). **Pola można ustawiać „naraz” albo każde oddzielnie.** Oprócz tego, również za pomocą enkapsulacji, klasa ma umożliwiać ustawienie parametru `zarobki netto`. Po ustawieniu zarobków, system ma automatycznie wyliczać możliwą wartość pożyczki dla klienta ze wzoru:

*pożyczka = (pensja / 7) * (65 - wiek)*

<u>Przykładowe działanie:</u>

```
Pożyczkobiorca Osoba1 = New Pożyczkobiorca();
Osoba1.setDane(„Jan”, „Kowalski”, 40, 40041416732, 4500); //w
     tym momencie ma się już wyliczać pożyczka
Osoba1.getPożyczka(); //wypisze wartość pożyczki na ekran
Pożyczkobiorca Osoba2 = New Pożyczkobiorca();
Osoba2.setImię(„Anna”);
Osoba2.setNazwisko(„Iksińska”);
Osoba2.setWiek(40);
Osoba2.setZarobki(10543); // w tym momencie ma się wyliczać
     pożyczka
```

Zad. 57:

Napisz klasę „generator losowych numerów telefonów”. Klasa ma umożliwiać generowanie podanej przez użytkownika liczby numerów telefonów z konkretnej strefy numeracyjnej. Przykładowe zastosowanie klasy `LosoweNumeryTelefonu`:

```
LosoweNumeryTelefonu randomTel = new LosoweNumeryTelefonu();

for(int i=11;i++;i<33) {

randomTel.setWojewództwo(i); //ustawienie strefy numeracji 32
randomTel.generate(1000); //wygenerowanie 1000 numerów,
     zapisywanych lokalnie w tablicy obiektu, a nie klasy

randomTel.save(i+„plik.tel”) //zapis do pliku tekstowego –
     ścieżka podawana w parametrze
}
```

Wynikiem działania powyższego kodu powinno być otrzymanie kilkunastu (ilu? - odpowiedz) plików z losowo wybranymi numerami telefonów (1000 sztuk) z poszczególnych stref numeracyjnych (11, 12, …, 32).
Skup się na w/w funkcjach, nie dodawaj dodatkowych funkcjonalności!

Zad. 58:

Napisz funkcję `zapisz jako()`, która będzie częścią klasy `Osoba`. Jej funkcjonalnością będzie zapisywanie informacji o osobach do pliku tekstowego. Załóż, że za każdym razem plik będzie miał nazwę złożoną z imienia i nazwiska danej osoby. Prototyp klasy `Osoba` przedstawia się następująco:

```
public class Osoba  {
     int wiek;
     String imię;
     String nazwisko;
     String[] dzieci = new String[10];
     double dochód;
}
```

Zad. 59:

Zaprojektuj dwupoziomową hierarchię klas i zapisz ją w pliku nagłówkowym:

1. Zaimplementuj klasę główną.
2. W klasie głównej zaimplementuj trzy konstruktory: bezparametrowy, parametrowy, kopiujący.
3. W klasie głównej zaimplementuj destruktor.
4. Klasa ma posiadać pola i metody prywatne i publiczne (co najmniej po jednym z każdego).
5. W klasie głównej zdefiniuj metodę wirtualną oraz niewirtualną. Nadpisz obie metody w klasie potomnej.
6. Zaimplementuj trzy klasy dziedziczące po klasie głównej.
7. W klasie głównej lub potomnej zdefiniuj składnik statyczny (np. zliczający liczbę obiektów danej klasy).
8. W klasie potomnej zaimplementuj przeciążenie trzech wybranych operatorów.
9. Dodaj, cokolwiek jeszcze ciekawego Ci przyjdzie do głowy :-)

W `main()` powołaj do życia kilka obiektów stworzonych klas. Pokaż na ich przykładzie działanie zaimplementowanych funkcjonalności z punktów 1-8.

`Figura`, podklasy: `Trojkat, Prostokat`, `Romb`

`Zwierze`, podklasy: `Pies`, `Kot`, `Chomik`

`Pies`, podklasy: `Jamnik`, `Terier`, `Husky.`

`Parzystokopytne`, podklasy: `Jelen`, `Zyrafa`, `Wielblad`

`Ptak`, podklasy: `Golab`, `Wrobel`, `Kos`

`Budynek`, podklasy: `Biblioteka`, `Urzad`, `Szpital`

`Mebel`, podklasy: `Szafa`, `Lozko`, `Krzeslo`

`Sztucce`, podklasy: `Widelec`, `Lyzka`, `Noz`

`CialoNiebieskie`, podklasy: `Gwiazda`, `Planeta`, `Satelita`

`Bog`, podklasy: `Zeus`, `Herkules`, `Apollo`

`Roslina`, podklasy: `Kwiat`, `Drzewo`, `Krzew`

`DzieloSztuki`, podklasy: `Obraz`, `Plakat`, `Rzezba`

`Przedmiot`, podklasy: `Informatyka`, `Fizyka`, `Matematyka`

`Pojazd`, podklasy: `Samochod`, `Autobus`, `Motor`

`Taniec`, podklasy: `ChaCha`, `Rumba`, `Walc`

`Plyta`, podklasy: `CD`, `DVD`, `BluRay`

`UrzadzenieMobilne`, podklasy: `Laptop`, `Smartfon`, `Notebook`